Brief Announcement: Hierarchical Consensus for Scalable Strong Consistency

Benjamin Bengfort and Pete Keleher

University of Maryland
{*bengfort,keleher*}*@cs.umd.edu*

Abstract

Distributed consensus algorithms are the primary mechanism of implementing strong consistency in geo-replicated data stores. Consistency is guaranteed by the explicit sequential ordering of accesses or locks and leader based consensus protocols minimize the number of messages required to apply a command without conflict detection in a fault-tolerant manner. However, distributed consensus algorithms must reject inconsistent updates to the system and as a result do not provide high availability or throughput. In this paper we describe *Hierarchical Consensus*, a new decentralized consensus protocol which distributes consensus decisions to multiple tiers of consensus groups according to shared accesses so as to maintain implicit dependencies. Hierarchical Consensus is unique in that it can efficiently guarantee the strict total ordering on replicated logs in the wider area even in the presence of partitions and varying connectivity. We will show its correctness and describe an experimental methodology that we will use to demonstrate durable, highly-available, and strongly consistent accesses in systems that scale from small to large networks.

1 Introduction

Strong consistency in a geo-replicated distributed data store requires a fault-tolerant mechanism that maintains consistency during node failure and communication partitions. Distributed consensus protocols inspired by Paxos [4] have been widely adopted to coordinate consistency, however, because of increased communication they cannot scale to arbitrary system sizes[2]. Several recent algorithms have attempted to address the scaling limitations of consensus and take two general forms — leader election and conflict detection.

Leader-oriented consensus such as MultiPaxos [4] and Raft [7] minimize the number of required communication phases by nominating a dedicated proposer. Less communication means better throughput, and fault-tolerance is achieved through node failure detection such as a heartbeat and a new leader is elected to minimize downtime. Leader-oriented approaches introduce two new problems, however: *load* and *distance.* Because the leader will necessarily do more work and handle more communication than other nodes, it must have sufficient resources to handle the workload; moreover, since any node can be elected leader, all nodes must have sufficient resources to handle the workload. This introduces scaling problems in two dimensions: adding nodes means more communication, increasing the minimum resource requirements for all nodes in the system. In geo-replicated systems, bandwidth and latency are highly variable therefore the election of a leader in a specific location means that consensus is bound by the slowest connection, making the consensus algorithm sensitive to distance. Although recent approaches such as S-Paxos [1] and Mencius [5] add load balancing to leader-oriented mechanisms, they cannot solve the distance problem.

Conflict detection approaches such as EPaxos [6] and MDCC [3] are optimistic that most consensus decisions are consistent. They propose “fast” and “slow” consensus paths, such that a subset of close nodes can quickly reach consensus but add dependency information to detect conflicts when commands are applied. If a conflict is detected, then decision must traverse the “slow” path to ensure correctness. Conflict detection does not have a distance problem, as nodes can select close neighbors, however this method does not guarantee dissemination of the command, which can require large amounts of dependency resolution. As the network scales, dependency graphs tend to increase in both size and complexity, increasing the load on the system.

In practice systems do not implement global consensus, but instead apply multiple consensus groups to coordinate specific objects or tablets. This keeps quorum sizes small, allocating just enough nodes to a quorum to maintain a minimum level of fault tolerance. However, in so doing, an object can only be consistent with respect to its own updates and the system loses information about dependencies. Moreover, there is no coordination between consensus groups, a single node can participate in multiple per-object consensus groups, which does not eliminate node and distance problems.

2 Hierarchical Consensus

Give concrete details and definitions.

2.1 Operation

2.2 Epochs and ordering

2.3 Correctness

3 Experimental Design

Discuss implementation of a file system. decoupling blob from version replication.

Three modes of operation

Forks, mapping hierarchical onto a file system.

4 Ongoing and Future Work

Current and future work.

References

[1] Martin Biely, Zoran Milosevic, Nuno Santos, and Andre Schiper. S-paxos: Offloading the leader for high throughput state machine replication. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, pages 111–120. IEEE, 2012.

[2] H. Howard, D. Malkhi, and A. Spiegelman. Flexible Paxos: Quorum intersection revisited. *ArXiv e-prints*, August 2016.

[3] Tim Kraska, Gene Pang, Michael J. Franklin, Samuel Madden, and Alan Fekete. MDCC: Multi-data center consistency. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 113–126. ACM, 2013.

[4] Leslie Lamport. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[5] Yanhua Mao, Flavio Paiva Junqueira, and Keith Marzullo. Mencius: Building efficient replicated state machines for WANs. In *OSDI*, volume 8, pages 369–384, 2008.

[6] Iulian Moraru, David G. Andersen, and Michael Kaminsky. There is more consensus in egalitarian parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 358–372. ACM, 2013.

[7] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, 2014.